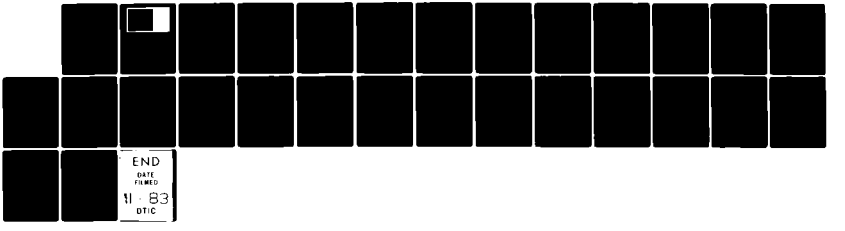
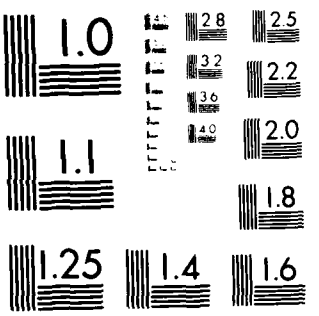
AD-A134 543 SOME EXPERIENCES WITH GASP-LIKE MICROCOMPUTER 1/1
SIMULATION LANGUAGES IN FOR..(U) WISCONSIN UNIV-MADISON
MATHEMATICS RESEARCH CENTER A THESEN ET AL. SEP 83
UNCLASSIFIED MRC-TSR-2571 DAAG29-80-C-0041 F/G 9/2 NL



END
DATE FILMED
11-83
DTIC



MRC Technical Summary Report #2571

SOME EXPERIENCES WITH GASP-LIKE MICROCOMPUTER SIMULATION LANGUAGES IN FORTRAN, PASCAL, AND ADA

Arne Thesen and Rekha De Silva

Mathematics Research Center
University of Wisconsin—Madison
610 Walnut Street
Madison, Wisconsin 53705

September 1983

(Received August 25, 1983)

Approved for public release
Distribution unlimited

Sponsored by

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park
North Carolina 27709

UNIVERSITY OF WISCONSIN - MADISON
MATHEMATICS RESEARCH CENTER

SOME EXPERIENCES WITH GASP-LIKE MICROCOMPUTER SIMULATION LANGUAGES IN FORTRAN, PASCAL, AND ADA

Arne Thesen and Rekha De Silva

Technical Summary Report #2571

September 1983

ABSTRACT

The modelling and programming philosophy behind micro-computer software used in teaching simulation is presented with implementations in FORTRAN-80, Pascal/MT+ and Janus (an Ada subset). The relative strengths and weaknesses of these implementations are discussed.

AMS (MOS) Subject Classification:

Key Words: Simulation languages, Microcomputers, performance evaluation.

Work Unit Number 6 - Miscellaneous Topics

A-1

Sponsored by the United States Army under Contract No. DAAG29-80-C-0041.

SIGNIFICANCE AND EXPLANATION

At the University of Wisconsin-Madison we have successfully used micro-computers for several years in our graduate level simulation courses. After some initial start up problems, mostly caused by our failure to realize that micro-computers are not used in the same manner as main frames, we feel that we are able to teach simulation at least as effectively with micros as we once were with our larger computers. One reason for this is our development of a simulation "language" specifically designed for use in an educational environment by "naive" users.

In spite of our success in using micro-computers for simulation for education, we have been having difficulties answering reasonable questions regarding the utility of micro-computer based simulations in non-educational settings. Typical concerns include issues such as:

1. Are micro-computers too slow for serious simulations?
2. Is sufficient memory available for large models?
3. Which programming language is best suited for model development?

In this paper we will attempt to provide answers to these questions.

To provide for a fair comparison between different programming languages we designed a simple discrete event simulation language with many of the modelling features of more elaborate languages. This language was then implemented in Microsoft FORTRAN, Pascal/MT+, and Ada/Janus, and the performance of these implementations were compared.

The target language is presented in Section II. The three implementations are discussed in Section III, and the evaluation of the implementation is presented in Section IV. The source codes of the three implementations are available in a separate document [12].

The responsibility for the wording and views expressed in this descriptive summary lies with MRC, and not with the authors of this report.

SOME EXPERIENCES WITH GASP-LIKE MICROCOMPUTER SIMULATION LANGUAGES IN FORTRAN, PASCAL, AND ADA

Arne Thesen and Rekha De Silva

I. INTRODUCTION

At the University of Wisconsin-Madison we have successfully used micro-computers for several years in our graduate level simulation courses. After some initial start up problems, mostly caused by our failure to realize that micro-computers are not used in the same manner as main frames, we feel that we are able to teach simulation at least as effectively with micros as we once were with our larger computers. One reason for this is our development of a simulation "language" specifically designed for use in an educational environment by "naive" users.

In spite of our success in using micro-computers for simulation for education, we have been having difficulties answering reasonable questions regarding the utility of micro-computer based simulations in non-educational settings. Typical concerns include issues such as:

1. Are micro-computers too slow for serious simulations?
2. Is sufficient memory available for large models?
3. Which programming language is best suited for model development?

In this paper we will attempt to provide answers to these questions.

To provide for a fair comparison between different programming languages we designed a simple discrete event simulation language with many of the modelling features of more elaborate languages. This language was then implemented in Microsoft FORTRAN, Pascal/MT+, and Ada/Janus, and the performance of these implementations were compared.

The target language is presented in Section II. The three implementations are discussed in Section III, and the evaluation of the implementations is presented in Section IV. The source codes of the three implementations are available in a separate document [12].

Sponsored by the United States Army under Contract No. DAAG29-80-C-0041.

II. THE TARGET LANGUAGE

A. Design Goals

The purpose of any simulation language is to free the analyst from coding tasks so that she can concentrate on model development and analysis. Here, this global goal was operationalized by specifying modelling capabilities, interface goals and resource restrictions as follows:

DESIGN OBJECTIVE
MODELLING CAPABILITIES (general purpose)
- Discrete event scheduling using user written event routines - Automatic scheduling of recurrent events - Floating point clock - Set modelling capabilities with automatic data collection - Service routines for collection and display of user specicfied data. - Keyboard control of running models. - Five common random number generators
USER INTERFACE (user friendly)
- The user should not need to understand the design of the simulation data base. - The user should not be able to cause run time errors in the simulation package (for example by removing a non existing entity). - The resulting code should be a self documented model - Host language should be designed to facilitate program debugging - Model verification should be facilitated by extensive error checking and an interactivly controlled trace feature.
RESOURCE CONSIDERATIONS (run on 8 bit micros)
- The language should be small enough to allow room for reasonably complex models on a 64k byte micro-computer. - Time required to compile and link a model should be sufficiently small to facilitate interactive model development on a CP/M based desk top computer. - Time to execute a model should not be excessive.

Table 1: Design Objectives.

The commands of a language intended to implement these objectives are listed in Table 2. To keep the size of the system small, some desirable features were omitted. For example:

- Model building blocks such as facilities and inventories are omitted.
- All sets are ranked on the first attribute value
- Entities can only be removed from the head of a set
- Simple linearly linked lists are used to represent all sets.
- Only the most common random number generators are provided.

Experience has shown that most commonly encountered model features can be easily programmed with the tools provided by this language. In the few cases where this was difficult, it was easy to add the required features.

EVENT SCHEDULING COMMANDS	Effect on simulation model
NextEvent	Updates simulation clock and global variables TNow and Code
Recurrent(EventCode,MeanInterval)	Schedules perpetual sequence of events of type EventCode at exponentially distributed random intervals with a mean of MeanInterval
Schedule (EventCode,TimeIncrement)	Schedules a single occurence of an event of type EventCode to take place TimeIncrment time units from current simulated time.

SET MANAGEMENT COMMANDS	Effect on the simulation model
InsertInto(SetNo,Attributes)	Inserts the entity described by the vector Attributes into the set SetNo , such that the first attribute value for set members are ranked in increasing order
RemoveFrom(SetNo,Attributes)	Removes the entity at the head of the set. Returns its attribute values in the vector Attributes
Size(SetNo)	Returns the count of entities currently in set SetNo

RANDOM NUMBER GENERATORS	Distribution	Type
Bernoulli(p)	Bernoulli with mean "p"	Integer
Exponential(tmean)	Exponential with mean "mean"	Real
Normal(mean,stdv)	Normal with mean "mean" and standard deviation "stdv"	Real
Poisson(lambda)	Poisson with mean "lambda"	Integer
Uniform(a,b)	Uniform between "a" and "b"	Real

OTHER COMMANDS	Function
Collect(bin, value)	Collect point values
Histogram(value)	Collect data for histogram
Initialize	Initialize entire data base
Report	Display statistical summaries
TimeIntegrate(Bin,NewValue)	Collect data for time integrated averages

Table 2: Commands Supported by Target Language

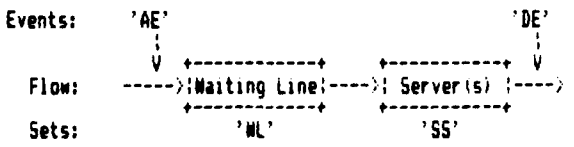
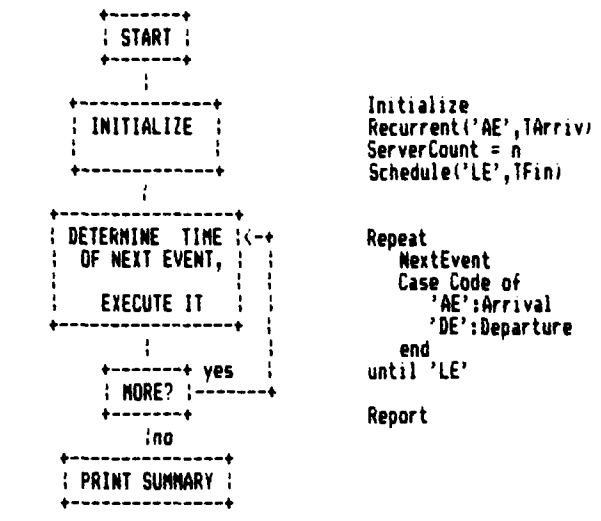


Figure 1: Process Model of N-Server Queuing system.

B. An Example

To introduce our simple language, we present in Program 1, a simulation model of the n-server queuing system described in Figure 1.




```
Initialize
Recurrent('AE',TArriv)
ServerCount = n
Schedule('LE',TFin)

Repeat
   NextEvent
   Case Code of
      'AE':Arrival
      'DE':Departure
   end
until 'LE'

Report
```

```
Procedure Arrival
begin
  If SizeOf('SS')<ServerCount
    then
      InsertInto('WL',TNow)
    else
      Serve(TNow)
end
```

```
Procedure Departure
begin
  RemoveFrom('SS',Eos,TEntry)
  Collect(2,TNow-TEntry)
   if  SizeOf('WL') >0  then
    RemoveFrom('WL',TEntry)
    Collect(1,Tnow-TEntry)
    Serve(TEntry)
  endif
end
```

```
Procedure Serve(EntryTime)
begin
  ServiceTime=Expo(MeanTime)
  InsertInto('SS',TNow+STime,EntryTime)
  Schedule('DE',ServiceTime)
end
```

Program 1: Target model of N-Server Queuing System

The overall logic of this model should be obvious and is not discussed here. Note that we chose to use an Algol-like block structure in this model. Obviously the program structure used in any implementation of the language is a function of the structures supported by the host language.

Statistics collected on sets Current time is = 100.00

Set No	Total Input	Current size	----Time-In-Set---- min	avg	max	Avg. size
SS	14	0	0.13	0.32	0.69	0.45
WL	4	0	0.06	0.12	0.25	0.05

Statistics on user collected data Current time is = 100.00

Unit	OBS	Mean	Var	StDev	Min	Max	
1	4	0.12	0.01	0.09	0.06	0.25	(Time in queue)
2	14	0.36	0.03	0.17	0.13	0.69	(Time in system)

Figure 2: Output from Program 1.

The statistical summary report produced by this model is presented in Figure 2, and a portion of the trace produced by the running model is reproduced in Figure 3. In addition to automatically collected data, Figure 2 also includes summaries on explicitly collected data on total time in the system and on time in queue.

Time	Command	Event	Set	n	Unit	Parameters	
0.0	Initialize						
	Recurrent	AE				0.49	0.50
	Schedule	LE				100.00	0.0
0.49	Event	AE					
	Recurrent	AE				0.71	0.50
	SizeOf		SS	0			
	InsertInto		SS	1		0.78	0.49
	Schedule	DE				0.78	0.29
0.71	Event	AE					
	Recurrent	AE				1.39	0.50
	SizeOf		SS	1			
	InsertInto		WL	1		0.71	
0.78	Event	DE					
	RemoveFrom		SS			0.78	0.49
	Collect				2	0.29	
	SizeOf		WL	1			
	RemoveFrom		WL			0.71	
	Collect				1	0.07	
	InsertInto		SS	1		1.10	0.71
	Schedule	DE				1.10	0.32

Figure 3: Partial trace produced by the model

C. MODELLING CONSTRUCTS

The language presented here is based on the GASP IV language [9]. Details of the language are discussed below.

1. Events and Event Scheduling

The act of entering information about future events in the future events list is referred to as **Event Scheduling.** A typical event list is shown in Figure 4. Two mechanisms for event scheduling are recognized. These are discussed below.

Time	Code	Events
203.54	'AE'	Arrival of customer
204.67	'DE'	Departure of customer
1000.00	'LE'	End of simulation

Figure 4: Typical event list.

A **recurrent** event stream is a sequence of events (such as customer arrivals) that occurs at random intervals independently of the state of the system at any point in time. These events usually describe external inputs to a model such as customers.

machine breakdowns, messages etc. In Program 1 the command Recurrent('AE',TArriv) is used to schedule a perpetual sequence of type 'AE' (arrival) events each of which is separated from its neighbor by an exponentially distributed random interval with a mean of TArriv.

A **scheduled** event is one that is explicitly inserted into the events list by the user written model. This modelling construct is used to create events that are triggered by state changes in the simulated system. In our example, the command Schedule('DE',expo(TServ)) schedules a 'DE' (departure) event to occur expo(TServ) time units from the current simulated time. Scheduled events differ from recurrent events in that they are usually created when the state of the system a certain specified state. Recurrent event streams *do not reflect* the state of the system.

2. User Defined Sets and Entities

The set modelling construct is used in most simulation languages to represent and hold entities as they flow through the system being simulated. In Program 1 we use the set 'WL' to hold the customers in the waiting line and the set 'SS' to hold customers being served.

In our simple language all sets are ranked in increasing order of the first attribute of each entity. In Program 1 we want to serve customers in the order they arrive. We therefore assign the time of arrival to attribute one for all customers entering the queue. Similarly, for customers entering the "being served set" we assign the end-of-service time to the first attribute.

Entities can only be removed from the head of a set. This is the most serious shortcoming of the present design. Our decision to exclude routines for removal of entities in the middle of a set was due to the increase in complexity to **find** the desired entity (the actual removal of an entity when its position is known is trivial).

D. DATABASE MANAGEMENT

To a user, a simulation language may be thought of as a modelling tool. To the language implementor however, it looks more like a data base management system. This is because an elaborate data base is required to keep track of the constantly changing state of the simulated system. In the following we discuss the conceptual design of this structure.

1. Events and Event Scheduling

Information about individual events are contained in **event notices**, each of which contain the following information:

1. Time of the event
2. Type of the event
3. Mean interval between recurrent events

To ensure proper execution of events, these event notices are stored (at least logically) in chronological order in the simulation data base. The task of maintaining the notices in this order is delegated to the event scheduling commands (Table 3).

Function	Effect on event set
NextEvent(EventCode,Time)	Retrieves code and time of next event in list and deletes the corresponding event notice
Recurrent(EventCode,MeanInterval)	Creates event notice. Inserts notice at proper place in set
Schedule (EventCode,TimeIncrement)	Creates event notice. Flags event as nonrecurrent. Insertes in set

Table 3: Effect on Event Scheduling Commands on the Database.

The design of efficient data structures for simulation has been a subject of considerable interest in recent years [2,5,7,13]. However to keep our language simple, the only data structure used in this project is the **linked list.** Figure 5 shows how the event set in Figure 4 can be represented as a linked list.

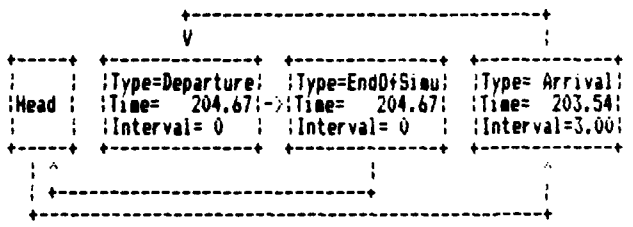


Figure 5: Linked List Representation of Events List in Figure 4.

Since the detailed implementation of data structures depends on the programming language used, we defer the specifications of record format and content for event notices to the three implemention sections.

2. User Defined Sets and Entities

Individual entities are represented by **entity records.** To keep the system simple, we are again using linked lists to maintain the records in proper order. To facilitate automatic collection of performance data, the header record for each different set includes space for the following statistical data:

1. Number of entries into the set.
2. Current size of the set.
3. Total time in set at time of last state change.
4. Time of last change in state of set.
5. Maximum time an entity has spent in the set.
6. Minimum time an entity has spent in the set.

In Table 4 we summarize the effect of different set management commands on the simulation data base.

Function	Effect on the data structure
InsertInto(SetNo,Attributes)	Obtains an entity record. Saves attribute values and other data. Links record into proper logical position in set.
RemoveFrom(SetNo,Attributes)	Moves attribute data from entity record to vector Attributes. Deletes entity record. Collects data on set utilization
Size(SetNo)	No effect on data structure

Table 4: Effect on Database of Set Management Commands

Language Features	Janus/Ada	FORTRAN-80	Pascal/MT+
Dynamic Memory Management	Yes	No	Yes
Include Files	Yes	No	Yes
Separately compiled library	Yes	Yes	Yes
Strongly Typed?	Yes	No	Yes
Variable Names	Any Length	6 char max	8 signif.chr
Variables			
Double Precision	No	Yes	No
Real Variables	No	Yes	Yes
Record Type	Yes	No	Yes
Scope of variabl.	Many options	Local,COMMON	Nested
Static Variables	Yes	No	No
Resource Reqmts			
Floppy Disks	2 @ 256k	2 @ 80k	2 @ 256k
Memory Required	64k	64k	64k
Operating System	CP/M	CP/M	CP/M

Table 5: Distinguishing Language Features.

III. IMPLEMENTATION

In this section we present the implementations of the target language in the three different programming languages. But first, for readers not familiar with the features of the languages used in this report, we present in Table 5 (page 8) a brief summary of important features.

A summary of the commands used in our implementations is given in Table 6. As we shall see, language differences have an important impact on the manner in which these commands are implemented.

Target Language	FORTRAN-80	Pascal/MT+	Ada/Janus
NextEvent	call NEXTEV(ievent)	NextEvent	Next_Event(Current_Event)
Recurrent(code,mean)	call RECUR(iev,tmean)	Recurrent(code,MeanInterval)	Recurrent(Event_Code,Mean)
Schedule(code,TIncr)	call SCHEDU(iev,tinter)	Schedule(code,intterval)	Schedule(Code,Tinter)
InsertInto(Set,Attr)	call INSERT(iset,a)	InsertInto(set)	Insert_Into(Set_number)
RemoveFrom(Set,Attr)	call REMOVE(iset,a)	RemoveFrom(set)	Remove_From(Set_Number)
Size(Set)	NSIZE(iset)	SizeOf(set)	Size_Of(Set_Number)
Collect(bin,value)	call COLCT(i,x) call ERROR(i)	Collect(BinNumber,Value)	Collect(Bin,Value)
Histogram(value)	call HISTO(X)	Histogram	
Initialize	call INIT	Initialize	Initialize_Events Initialize_Sets Initialize_Data_Collection
Report	call REPORT	Report	Set_Report Collect_Report
TimeIntegrate(bin,V)	call TCOLCT(i,xt)	Tcollect (Bin,Data)	TCollect(Bin,Value)
Bernoulli(p)	IBERNO(p)	Bernoulli(p)	
Exponential(tmean)	EXPO(t)	Expo(dmean)	
Normal(mean,stdv)			
Poisson(lambda)	IPOISN(a)	Poisson(lambda)	
Uniform(a,b)	UNIF(A,B)	Unif(llimit,ulimit)	Random(A,B)

Table 6: Summary of Implemented Commands

A. FORTRAN-80

1. Background

The FORTRAN implementation is written in Microsoft FORTRAN-80 [8]. This FORTRAN dialect has ANSII FORTRAN IV as a subset. (Unusual extentions include the ability to give identical names to variables, subroutines and COMMON areas.) The compiler and linker are both fairly small. In fact this is the only language that we can use on our Heath/Zenith Z89 micro-computer with single density 5 1/4" disks. The language has some annoying idiosyncracies. The more important of these are mentioned in Section IV.

2. The N-Server Queuing System

To illustrate the FORTRAN implementation we present in Program 2 the FORTRAN version of the N-server queuing system example.

```
C +-----------------------------------------+
C ! SIMULATION MODEL OF SIMPLE QUEUING SYSTEM !
C +-----------------------------------------+
      common /data/nserv,tarriv,tserv,ibin
      common /CLOCK/tnow,tend
      common /COLCT/nc(10),other(40)
      common /LIST/nlists,ievset,ighead,alist(4,400),iptr(400)
      common /SEED/iseed
      common /TRACE/itrace
C VARIABLES
C ---------
C       nserv  = number of servers
C       tarriv = mean inter arrival time
C       tserv  = mean service time
C
C MODELLING UNITS
C ---------------
C       Event(1) = Customer arrival  (recurrent event)
C            (2) = End of service    (triggered event)
C       Colct(1) = Wait time in queue
C            (2) = Wait times in system
C
      call INIT
      tarriv = 0.5
      TFin   = 100.0
      nserv  = 2
      tserv  = 0.8
      call RECUR(1,tarriv)
      call SCHED(3,TFin)
 1000 call NEXTEV(icode)
        if(icode.eq.1) call arrvl
        if(icode.eq.2) call depart
        if(icode.ne.3) goto 1000
      call REPORT
      stop
      end
C     ---------------
      subroutine arrvl
C     ---------------
C  place in service if server is available
      dimension attrib(4)
      common/data/nserv,tarriv,tserv,ibin
      common/CLOCK/ tnow
      if  (NSIZE(1).ge.nserv) go to 1000
      call serve(tnow)
      return
C     place in queue
 1000 attrib(1) = Tnow
      attrib(2) = Tnow
      call INSERT(2,attrib)
C  schedule next arrival
      return
      end
C     ------------------------
      subroutine serve(tentry)
C     ------------------------
      dimension attrib(4)
      common /data/nserv,tarriv,tserv,ibin
      common /CLOCK/tnow
      twait=tnow-tentry
      call COLCT(1,twait)
      ts=ERLANG(tserv,3)
      attrib(1)=ts+tnow
      call INSERT(1,attrib)
      call SCHED(2,ts)
      return
      end
C     -----------------
      subroutine depart
C     -----------------
      dimension attrib
      common /data/nserv,tarriv,tserv,ibin
      common /CLOCK/tnow
C  server finishing first is first in set
      call REMOVE(1,attrib)
      tsys=tnow-attrib(2)
      call COLCT(2,tsys)
      if(NSIZE(2).le.0) return
      call REMOVE(2,attrib)
      call serve(attrib(1))
      return
      end
```

Program 2: FORTRAN Version of N-Server Queuing System

3. The Data Base System

Our FORTRAN-80 implementation is particularly weak in the set management area. Since FORTRAN IV does not support records and dynamic memory management, it was necessary to create a large two dimensional array, each row of which would be available for possible use either as an event or as an entity record. (Therefore both record types are of the same size.) When not in use, these rows are strung together into a **garbage list** from which records are drawn as needed, and to which records are returned when no longer in use. The following subroutines and functions were developed for this purpose:

Subroutine/Function	Purpose
INIT	Creates the record pool and links all records into the garbage list.
NEWREC(dummy)	Removes next available record from the garbage list and returns its index
LNKREC(iset,irec)	Links record 'irec' into set number 'iset'
LNKOUT(iset,ipos)	Removes the record in logical position 'ipos' from set 'iset', returns its index

Table 7: List Processing Routines for FORTRAN Implementation.

The records used for event scheduling and set management are all stored in the array LIST in the common area /LIST/ (note the non standard naming convention). Their content is described in Figure 6. Note that a considerable amount of memory space is wasted due to the restriction that all records be of the same size.

Field number	HEADER RECORDS: USER DEFINED SET HEADER	HEADER RECORDS: EVENT NOTICE HEADER	ENTITY RECORDS: USER DEFINED ENTITY	ENTITY RECORDS: EVENT NOTICE
1	Number of insertions	Number of insertions	attrib(1)	Time of event
2	Current size of set ranked	Current size of set ranked		
3	Time spent in set by entities no longer in set	not used	attrib(2)	Event Code
4				
5	Time of last change in state of set	not used	attrib(3)	not used
6				
7	not used	not used	attrib(4)	Avg interval for recur'nt events, or zero
8	not used			

Figure 6: Content of Records in FORTRAN-80 Implementation

For convenience we store 4 byte reals and 2 byte integers in the same array. The following equivalence statements in FORTRAN are used for this purpose:

```
REAL*4 ALIST(4,100)
INTEGER*2 LIST(8,100)
EQUIVALENCE ( ALIST(1,1),LIST(1,1) )
```

All set related data is stored in the COMMON area /LIST/. Headers, event records, and entity records are stored in the array **ALIST**; pointers are stored in the array **IPTR**. In Figure 7 we illustrate how the event list in Figures 4 and 5 might be stored in this structure:

i	1 Bar bag	2 Fut ure	3 set 1	4 set 2	5	6	7	::	99	record number
list(1,i)	0	-	0	0				99	1	
list(2,i)	-	3	-	-	204.67	1000.0	203.54	-	-	
list(3,i)	-	-	-	-	2	3	1	-	-	
list(4,i)	-	-	-	-	-	-	-	-	-	
alist(3,i)	-	-	-	-	-	-	-	-	-	
alist(4,i)	-	-	-	-	0.00	0.00	3.00	-	-	
iptr(i)	8	7	3	4	6	2	5	3	1	

Figure 7: FORTRAN Representation of Event List in Figure 6.

4. Some Notes on Program Development

In developing the target language we took special care to separate information that the modeller needs from information that she does not need. This design goal was only partially met in our FORTRAN implementation. Specifically, we were unable to hide the simulation data base from the user, as she is required to include data base related COMMON statements in almost all her subroutines. Not only does this reduce program readability, it introduces a significant source of errors as programmers frequently fail to place identical COMMON statements in all functions and subroutines.

Another goal of the target language was to facilitate the development of models that were self documenting and easy to read. Again, our FORTRAN implementation was only partially successful. With short variable names, no compile time constants and no block structure, it is extremely difficult to write readable complex programs in FORTRAN.

B. PASCAL/MT+

1. Background

Pascal/MT+ is a CP/M based true compiler with useful extensions such as overlays, strings and separately compiled modules. This compiler is substantially larger and slower than our FORTRAN compiler, and we were not able to use it on our Heath/Zenith Z89 computer with single density 5 1/4" drives. Instead, we used a larger system (Sierra Data Systems) with 8" drives.

2. The N-server queuing system

Our Pascal implementation of the N-server queuing model is presented in Program 3.

```
program qi(input,output);
{$IDecIns}
{$IDecII}
        {Start of user written model}
{-------------------}
 procedure Simulate;
{-------------------}
 var
{-----variables used in simulation model-----}
   EndTime             : Real;
   EosTime             : Real;
   MeanInterArrivalTime : real;
   MeanServTime        : Real;
   Servers             : Integer;
{----------------}
 procedure arrival;
{----------------}
begin
  attribute[1]:=TNow;
  if SizeOf(BeingServedSet) < Servers
    then StartService
  else InsertInto(WaitingLineSet);
end;
{----------------------}
 procedure StartService;
{----------------------}
var
  ServiceTime : real;
  TimeWaited  : real;
begin
  TimeWaited := TNow - ate[2];
  if ( TimeWaited <> 0.0
    Collect (1,TimeWaited
  ServiceTime:= Erlang(MeanServTime,3) ;
  Attribute[1]:=ServiceTime + TNow;
  InsertInto(BeingServedSet);
  Schedule(DepartureEvent,ServiceTime);
end;

{---------------}
 procedure Depart
{---------------}

 var
   TimeInSystem : real;

 begin
   RemoveFrom(BeingServedSet);
   TimeInSystem := Tnow - Attribute[2];
   Collect ( 2, TimeInSystem );
   If SizeOf(WaitingLineSet) > 0 then begin
      RemoveFrom (WaitingLineSet);
      StartService;
   end;
 end;


 begin                (Simulate)

   EndTime := 10;
   MeanInterArrivalTime := 0.5;
   MeanServTime := 0.4;
   Servers := 1;

   Initialize;
   Recurrent (ArrivalEvent, MeanInterArrivalTime);
   Schedule  (LastEvent, EndTime);

   repeat
     NextEvent;
     case CurrentEvent of
         ArrivalEvent   : arrival;
         DepartureEvent : depart;
     end;
   until (CurrentEvent = LastEvent);
   Report;
   Histogram;
 end;              (Simulate)
             ( end of user model )
 begin
    Simulate;
 end.
```

Program 3: Pascal/MT+ version of the N-server queuing model

3. Data base Design

The design of the database capitalizes on the fact that Pascal is able to create and dispose of records dynamically as needed without any cumbersome user-defined garbage list. (The price we pay for this is that we no longer know in advance the exact location of any specific record. Instead, we use pointer variables to lead us indirectly to the desired information.)

Event notices are represented as records with the following format:

```
Eventnotice : record
    Time  : real;          (Time of event)
    Type  : integer;       (Type of event)
    Next  : ^ Eventnotice  (Pointer to another event notice)
end;
```

Entities are represented as records with an essentially similar format.

Unlike our FORTRAN implementation in which all records were required to be of the same size, our Pascal implementation allows records of different sizes. We exploit this flexibility by making our header records larger than our entity records so that additional performance information can be collected for each set.

The structure of the resulting database is presented in Figure 8:

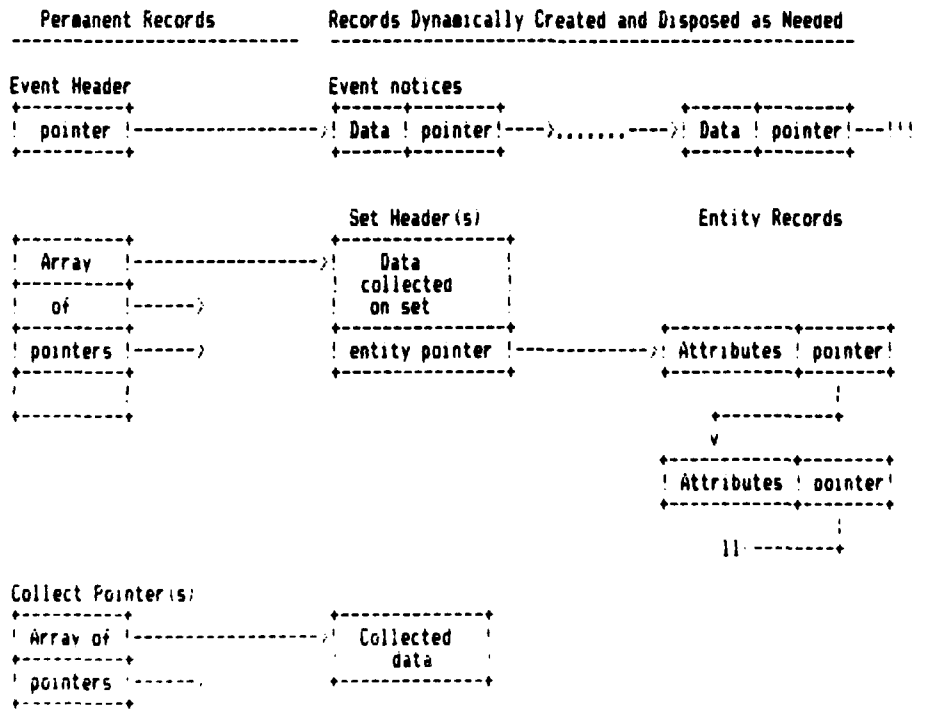


Figure 8: Structure of the Pascal Database

4. Some Notes on Program Development

Pascal is a strongly typed language (i.e. all variables must be defined before they are used). Students trained in FORTRAN find this to be annoying at first. However, there is no doubt that this feature substantially reduces the length of time it takes to develop a working program.

Since Pascal does not support static variables, it was necessary to give a global scope to all variables used by our language. This was done by declaring these variables in the main program. Fortunately, Pascal/MT+ provides an "include" macro instruction whereby the name of a file containing declarations can be used in lieu of the declarations themselves. This reduces user effort and increases readability.

A separate procedure "Simulate" is used to contain the user written model. This causes a clear separation between user written code and data and global data used by the language itself.

C. ADA/JANUS

1. Background

Ada is a large and complex language that appears from its specifications to be extremely well suited for the design goals of this project. A nice introduction to Ada is given in [1]. Bryant [3] discusses simulation programming in Ada.

To our knowledge no complete implementation of Ada is available for CP/M based micro-computers. However several "subsets" are available. We acquired two versions for this study. Unfortunately neither of these provide all the features required for a simulation language. Ada/Supersoft [11] does not support records and dynamic memory management (not to mention packages and separate compilation). Ada/Janus [10] provides these features, but does not (yet) support real variables. Both vendors promise to implement the entire Ada languages in later versions.

In order to investigate the powers of Ada, we chose to use Ada/Janus, as we felt that the omission of real variables is outweighed by the presence of records, memory management and separately compiled packages. (We understand that real variables will be provided "soon".)

At a first glance one may notice that Ada is very similar to Pascal. However, Ada has more to offer in elegance and program structure. Ada boasts of a high level of data abstraction. By this we mean that it is possible in ADA to build modules which are entirely independent of each other, and which can be used without any knowledge of the internal design of the module. This is one of the most advanced features of ADA.

These modules are called "Packages". Packages are the main constructs of an ADA program. Each package has a declaration part (optional) and a body part. The declaration part represents the resources (data structures, types and the names of the procedures/functions) available to the user. The package body implements these resources. The package body may contain implementation-dependent resources hidden from the user. Packages containing only type and object declarations do not need a body, since there are no implementation details to hide from the user. These may be used like the named COMMON blocks in FORTRAN, only these can be used much more flexibly.

Unlike Pascal, where all the global TYPES, CONSTANTS, data structures and variables had to be declared in the beginning of the program, and shared by all the modules, ADA can build up packages such that the same global TYPES, CONSTANTS, data structures and variables are distributed among these packages to define logically related collections of resources. At compile time these packages act like separate library routines which could be seen and used by other packages.

2. The N-server queueing system

The Ada implementation of the N-server queueing model is as follows:

```
With s,Events, Sets, Datacol;

package body Simulate is
use s, Events, Sets, Datacol;

   Arrival_Event   : constant := 'a';
   Departure_Event : constant := 'b';
   Last_Event      : constant := '1';

   BeingservedSet  : constant := 1;
   WaitingLineSet  : constant := 2;

   Current_Event   : character;
   End_Time        : integer := 32000;
   Mean_Inter_Arrival_Time : integer := 8;
   Mean_Service_Time       : integer := 10;
   Number_of_Servers       : integer := 3;


-------------------------
procedure Start_Service is
-------------------------
  Service_Time : integer;
  Time_Waited  : integer;

   begin
     Time_Waited  := Time_Now - Attribute (2);
     if Time_Waited > 0 then
       Collect (1,Time_Waited);
     end if;

     Service_Time := Random (Mean_Service_Time);
     Attribute(1) := Service_Time + Time_Now;
     Insert_Into (BeingServedSet);
     Schedule (Departure_Event, Service_Time);
  end;
```

```
-------------------
procedure arrive is
-------------------
  begin
    attribute(1) := Time_Now;
    attribute(2) := Time_Now;
    if Size_of(Beingservedset) < Number_of_Servers
        then Start_Service;
      else Insert_Into (WaitingLineSet);
    end if;
  end;

-------------------
procedure depart is
---  -------------
  Time_Spent_in_System : integer;

  begin
    Remove_From(BeingservedSet);
    Time_Spent_in_System := Time_Now - Attribute(2);
    Collect ( 2, Time_Spent_in_System );
    if Size_of (WaitingLineSet) > 0 then
       Remove_From (WaitingLineSet);
       Start_Service;
    end if;
  end;


  begin
    Initialize_Event_List;
    Initialize_Sets;
    Initialize_Data_Collection;
    recurrent(Arrival_Event, Mean_Inter_Arrival_Time);
    Schedule (Last_Event, End_Time);

    loop
      Next_Event (Current_Event);
      if (Current_Event=Last_Event) or else (Time_Now>32760)
          then exit;
        elsif Current_Event = Arrival_Event then arrive;
        else depart;
      end if;
    end loop;

    Set_Report;
    Collect_Report;

  end simulate;
```

Program 4: ADA/Janus N-Server Queueing Model

3. Database Design

The data structures used in the ADA/Janus implementation is identical to that of the Pascal MT+ implementation. However, in this implementation these structures are maintained in different packages; i.e. the set management package initiates and maintains the set header records and the entity list, the event scheduling package initiates and maintains the event list, etc.

The scope of the variables of a package is only within

that package itself. However, a package can be made visible to another at compile time with the use of a "WITH" clause. This means that one package can use and alter another's data, and local variables of a package will remain in existence between calls to the package.

4. Notes on Program Development

In our ADA/Janus implementation we found it easy to develop and hide from the user separately compiled modules each containing parts of the entire simulation database and list processing routines. This was one of our major design goals which we were able to achieve only in this implementation. Another goal we achieved here was total freedom in the length of variable names. This also contributed to the code being extremely self-documenting.

Ada's modules are useful tools for providing the programmer with modular computational ease. They allow algorithms /operations to be independently developed and used as components of larger programs. This facilitates top-down program development which is only possible with a language whose program units have a well defined user interface that hides the implementation details. The only global variables in our implementation were time "Time" and "Trace".

The packages we developed in addition to the main program were:

Event scheduling package (EVENTS):
- contains in its declaration part the event record type, event list, etc. It contains in its body the procedures for event scheduling given in Table 3. The random number generator also resides in this package body since it was used for the generation of events.

Set management (SETS):
- contains in its declaration the structure of the set header records, entity records, etc., and in its body the procedures outlined in Table 4.

Data Collection (DATACOL):
- contains in its declaration the data structures of the various data collecting arrays, and its body can be seen in [12].

A minor drawback of this implementation is that the file containing the package had to have the same name as the package. This restricted the package name to 8 characters. Vendors of Ada/Janus hope to have this restriction removed soon.

IV. EVALUATION

Given the many different schools of thought in all areas of applied simulation analysis, we do not hope to be able to provide definitive answers to the question of which language is "best" for simulation. Instead, we restrict our evaluation to a comparative study our three micro-computer bases language implementations. The results of this evaluation are discussed in the following sections, and they are summarized in Table 8.

DESIGN OBJECTIVE	Objective met ?		
MODELLING CAPABILITIES (general purpose)	Ada/Janus	FORTRAN-80	Pascal/MT+
- Discrete event scheduling using user written event routines	yes	yes	yes
- Automatic scheduling of recurrent events	yes	yes	yes
- Floating point clock	no	yes	no
- Set modelling capabilities with automatic data collection	yes	yes	yes
- Service routines for collection and display of user specicfied data.	only integers	yes	yes
- Keyboard control of running models.	yes	yes	yes
- Five common random number generators	only integers	yes	yes
USER INTERFACE (user friendly)			
- The user should not need to understand the design of the simulation data base.	yes	no	yes
- The user should not be able to cause run time errors in the simulation package (for example by removing a non existing entity).	yes	somewhat	yes
- The resulting code should be a self-documented model	yes	no	yes
- Host language should be designed to facilitate program debugging	yes	no	yes
- Model verification should be facilitated by extensive error checking and an interactively controlled trace feature.	yes	yes	yes
RESOURCE CONSIDERATIONS (run on 8 bit micros)			
- The language should be small enough to allow room for reasonably complex models on a 64k byte micro-computer.	yes	yes	yes
- Time required to compile and link a model should be sufficiently small to facilitate interactive model developement on a desk top computer.	perhaps	yes	yes
- Time to execute a model should not be excessive.	perhaps	perhaps	perhaps

Table 8: Evaluation summary.

A. Modelling Capabilities.

With the exception of Ada/Janus' absence of floating point variables, we were able to implement the entire simulation system in each programming language. However, the FORTRAN implementation was less flexible than the other implementations as we were required to specify at compile time the maximum dimensions of all data collection and set management arrays.

Each language provided the necessary operating system interface routines to facilitate real time interruption of a running simulation model.

B. User interface.

Perhaps the most significant differences in our implementations are in the data base area. Our primary goal of hiding the data-base from the user was only fully met in Ada/Janus where the use of separate packages completely removed the entire simulation database from the scope of the user. We were almost successful in hiding the database in Pascal/MT+. By nesting the user programs inside the simulation system, we relieved the user from having to define the database. However the entire database is still within her scope. In FORTRAN-80 it was not possible to hide the database at all.

The related goal of preventing the user from accidentally changing the content of the database was met in Ada/Janus and Pascal. It was only partially met in FORTRAN-80. (Errors in user written COMMON statements might result in a corrupted data base.)

With respect to the goal that the languages result in readable simulation models, we would judge that this goal was reached for Ada/Janus and, to a slightly lesser degree by Pascal/MT+. The goal was not reached for the FORTRAN-80 implementation. (A seen in Program 2, even a simple, well written FORTRAN-80 program is difficult to read.) These differences in readability were caused by several factors. We believe the most important of these to be:

1. The ability to hide confusing details
2. The use of long variable names
3. Block structured programming (begin-end, if-then-else)
4. Use of named constants.
5. Use of records.

We were able to implement reasonable error checking and trace features in all systems. Modelling errors should therefore be equally easy to find in each implementation. This is not so, for coding mistakes. Pascal/MT+ and Ada/Janus are both strongly typed languages. This means that each variable must be explicitly defined before it is used. While this is annoying to a FORTRAN trained programmer, there is no doubt in our opinion that this is a valuable feature that helps eliminate coding errors and reduce the total time required to develop a working program.

FORTRAN-80's error messages were confusing and often misleading. In addition, the FORTRAN-80 compiler proved to have a few strange bugs. For example blank lines at some times cause unpredictable (at least to us) errors, at other times they were accepted without complaints. More seriously, there appears to be a bug in the way arguments are passed to and from subroutines. Passing of constants sometimes caused unpredictable values to be returned. Finally the FORTRAN-80 manual is about as readable as the old IBM FORTRAN IV manual.

Pascal/MT+ is much easier to debug and has much better compiler error messages. The manual is also much better. (However we feel that the inclusion of more complete examples would be helpful). Ada/Janus has all the advantages over FORTRAN that Pascal has, and in addition its manual is quite readable.

C. Resource Requirements

1. Memory Requirement

As presented in Table 9 significant differences in storage requirements were observed:

	Ada/Janus	FORTRAN-80	Pascal/MT+
Size of File containing Executable Model	26k	44k	27k
Maximum Queue size for N Server model	2035	1562	1007

Table 9: A Comparison of Memory Requirements

It is seen that a FORTRAN-80 model requires substantially more disk space than the other languages. While this in itself is not a serious problem, it is a symptom of a serious drawback with the FORTRAN-80 system; namely that the linker creates a file that contains both the executable code and the data storage area. Since the linker and this file must be memory resident at the same time, the effective memory space available to a program is reduced. (The other compilers are able to store data in the area used by the linker, FORTRAN is not).

In order to get a measure of the relative size of problems that could be modelled with our three implementations, we measured the largest queue sizes that could be accomodated without memory overflow.

This result is also presented in the above table. It is seen that Pascal/MT+ could handle about 1000 customers at one time while FORTRAN could handle 1500 and Ada/Janus about 2000. Since Fortran is a much smaller language than Pascal we were not suprised to see that it left more space for data. Note however that whenever we changed dimensions in our FORTRAN database we had to recompile and link the entire program. This was not necessary in Pascal and ADA/Janus. The fact that Ada/Janus left even more space than Pascal was quite surprising until we realized that the records used to represent event notices and set members in Ada are smaller as two byte integers were used instead of the four byte real variables used in the other languages. Adjusting for this fact, the Ada/Janus model is not as memory efficient as the FORTRAN-80 model but slightly more efficient than the Pascal/MT+ model.

2. Speed

In Table 10 we present the results of four different timing evaluations. (A fifth measure "time to simulate 1000 departures" yielded meaningless results due to Ada/Janus' absence of floating point variables).

	Ada/Janus	FORTRAN-80	Pascal/MT+
Time to compile Simulation library	795 sec	75 sec	62 sec
Time to compile model	116 sec	16 sec	62 sec
Time to link program	53 sec	88 sec	56 sec
Time to compile and link	169 sec	104 sec	108 sec

Table 10: A Comparison of Resource Requirements

It is seen that the time to compile the simulation system itself is an order of magnitude slower with Ada/Janus than with the other systems. This is not a serious problem as the user would not be expected to recompile this system very often. Turning now to the time required to compile the model and to link it to the rest of the simulation system, we see that FORTRAN has the fastest compiler and the slowest linker. Combining the times for compiling and linking, we see that Pascal/MT+ ,and FORTRAN-80 are equally fast (or slow?) and that Ada/Janus is about 50% slower.

V. CONCLUSION

Each of our language implementations excel in some areas. FORTRAN-80 is fast and memory efficient, Pascal/MT+ is user friendly and reasonably fast, and, Ada/Janus has an exceptionally nice structure leading to programs that are quite readable and easy to work with.

On the other hand FORTRAN-80 is difficult to debug, Pascal/MT+ is not very memory efficient , and Ada is slow and does not at this time support real variables.

Given the absence of real variables in the present implementation of Ada/Janus, we conclude that of the three languages, Pascal/MT+ is the one best suited for microcomputer based simulation analysis at this time. However we were extremely impressed with the structure of Ada/Janus, and, when floating point variables become available, we believe Ada/Janus would be the language worth considering.

As a final consideration, we note that the close to two minutes required to compile and link our simulation models felt quite excessive. Since most models are compiled and recompiled a large number of times, this may indeed be prohibitively excessive. We therefore believe that other approaches to micro-computer based simulation analysis that do not involve compilation and linking of user written programs are likely emerge in the near future as a more viable approach to micro computer based simulation modelling.

BIBLIOGRAPHY

1. Barnes, J.G.P., **Programming in Ada**, Addison-Wesley, 1982.

2. Blackstone, John H., Gary L. Hogg, and Don T. Phillips, A Two-List Synchronization Procedure for Discrete Event Simulation. **Communications of the ACM**, Dec 1981, Vol 24, No. 12, pp824-829.

3. Bryant, Ray, Discrete Systems Simulation in Ada, **Simulation**, Oct 1982, pp111-121.

4. Digital Research ,**Pascal/MT+ User's Guide Release 5**, Fifth Printing, Pacific Grove, CA, 1981.

5. Franta, W.R., and Kurt Maly, A Comparison of Heaps and the TL Structure for the Simulation Event Set, **Communications of the ACM**, October 1978, Vol 21, No 10, pp 873-875.

6. Grogono,Peter. **Programming in Pascal**,, Addison-Wesley, Reading, Mass, 1980.

7. McCormack, William M. and Robert G. Sargent, Analysis of Future Eventset Algorithms for Discrete Event Simulation, **Communications of the ACM**, December 1981, Vol. 24, No 12, pp801-812.

8. Microsoft, **Fortran-80 Documentation**, Bellevue, WA, 1979.

9. Pritsker, A.A.B. , **The GASP IV Simulation Language**, Wiley, New York, 1974.

10. RR Software, **Janus/Ada Package User Manuals**, Madison WI, 1983.

11. SuperSoft Inc. **Ada Release 1.00a User's Manual**, Champaign, IL, 1982.

12. Thesen,Arne and Rekha Desilva, **S.ADA, S.For, and S.PAS Program Listings**, Department of Industrial Engineering, University of Wisconsin-Madison, 1983.

13. Vaucher, Jean G. and Pierrie Duval, A Comparison of Simulation Event List Algorithms, **Communications ot the ACM, April 1975, Vol 18, No 2, pp223-233.**

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER #2571	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Some Experiences with GASP-like Microcomputer Simulation Languages in FORTRAN, Pascal, and Ada		5. TYPE OF REPORT & PERIOD COVERED Summary Report - no specific reporting period 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Arne Thesen and Rekha De Silva		8. CONTRACT OR GRANT NUMBER(s) DAAG29-80-C-0041
9. PERFORMING ORGANIZATION NAME AND ADDRESS Mathematics Research Center, University of Wisconsin 610 Walnut Street Madison, Wisconsin 53706		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Work Unit Number 6 - Miscellaneous Topics
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Research Office P.O. Box 12211 Research Triangle Park, North Carolina 27709		12. REPORT DATE September 1983 13. NUMBER OF PAGES 24
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Simulation languages, Microcomputers, performance evaluation.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The modelling and programming philosophy behind micro-computer software used in teaching simulation is presented with implementations in FORTRAN-80, Pascal/MT+ and Janus (an Ada subset). The relative strengths and weaknesses of these implementations are discussed.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)